DS_Project_1 보고서

2023202027 정정윤

Introduction

자막 관리 프로그램을 제작하는 프로젝트로, 큐, 연결 리스트, 이진 탐색 트리를 사용하여 구현하여야 하는 프로젝트이다.

자막 관리는 명령어를 통해 이루어지며, 명령어와 출력은 파일입출력을 통해 특정 파일로부터 입력받고, 특정 파일을 통해 출력해야 한다.

매 명령어는 한 줄로 구성되며, 명령어의 인수의 적합성을 평가하여, 적합한 경우 명령이 작동하고, 그렇지 않은 경우 오류 코드를 출력해야 한다.

LOAD 명령어는 다른 인수를 받지 않고, 자막 데이터가 저장된 파일로부터, 프로그램 내부의 큐로 자막 데이터를 저장한다.

QPOP 명령어는 다른 인수를 받지 않고, 프로그램 내부의 큐에서 프로그램 내부의 이진 탐색 트리로 자막 데이터를 옮긴다.

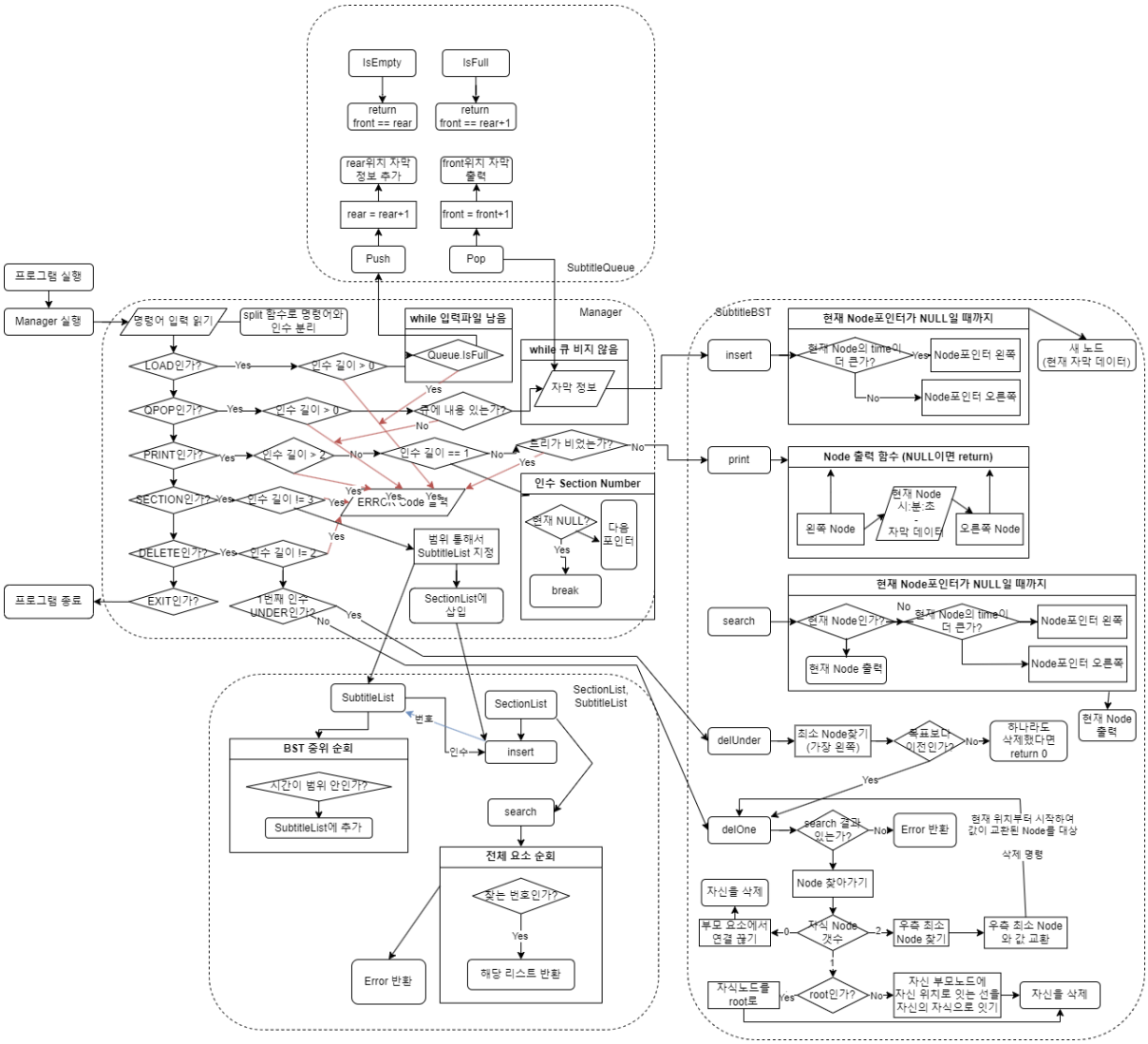
PRINT 명령어는 인수를 받지 않거나 1개 받을 수 있고, 받은 경우 프로그램 내부의 연결 리스트에 저장된 섹션중, 인수로 받은 수를 섹션 번호로 갖는 섹션 내의 자막 데이터를 출력한다.
받지 않은 경우, 이진 탐색 트리에 저장된 모든 자막 데이터를 출력한다.

SECTION 명령어는 인수를 3개 받으며, 첫 번째 인수는 섹션 번호, 두 번째 인수는 시작 시간, 세 번째 인수는 종료 시간이다. 시간 시간 이후 종료 시간 이전의 모든 자막 데이터를 이진 탐색 트리로부터 복사하여 자막 연결 리스트를 만든 후 섹션 연결 리스트에 섹션 번호와 함께 저장한다. 만약 저장할 자막 데이터가 하나도 없다면 오류를 반환한다.

DELETE 명령어는 인수를 2개 받으며, 인수는 삭제 설정으로 첫 번째로 UNDER를 받게 되면, 두 번째 인수로 받은 시간 이전의 모든 자막 데이터를 삭제하고, EQUAL을 받게 되면 두 번재로 인수로 받은 시간에 해당하는 자막 데이터를 삭제한다. 삭제할 데이터가 없었을 경우 오류를 반환한다.

구현 시에는 미리 일부 작성된 코드 파일을 바탕으로 작성하며, 기존에 작성된 코드에 있는 클래스명, 함수명은 임의로 변경할 수 없다. 추가로 클래스 및 함수를 구현하는 것에 제한은 없다.

FlowChart



[그림 1] 전체 순서도 그림

(https://drive.google.com/file/d/1CWlzLXQKy0uiohYfkhtD1yxT73E6j-nq/view?usp=sharing)

프로그램의 전체적인 객체 이용에 따른 순서도로 표현하였다. 왼쪽의 프로그램 실행과 Manager 실행을 기준으로 종료는 Manager객체와 이어지도록 그렸다.

Manager객체 주변으로 SubtitleQueue, SubtitleBST, SectionList(SubtitleList) 객체의 기능들을 근처에 묶어두어, Manager객체 내부에 있는 이진 탐색 트리와 큐, 연결 리스트에 대한 명령 사용을 근처에 있는 클래스의 함수를 사용하는 것으로 볼 수 있도록 하였다.

조건 표현 또는 실행 표현은 상세한 로직보다는 단순한 표현 위주로 작성하였다. 단지 로직이 중요한 부분은 while Container 등을 이용한 표현을 사용하였다.

교차를 하지 않게 되면 길게 돌아가야 하는 부분은 직관을 위해 색이 다른 선을 이용하여 교차하여 선이 돌아가지 않도록 하면서도 직관성을 유지하였다.

Manager의 명령 처리 과정에 대해서 각 명령어 비교 구문을 수행한다. 각 명령어는 수행됨에 따라 먼저 인수 길이의 확인을 수행하게 된다. 프로그램 작성 상세에서 인수가 올바른 개수만큼 들어오지 않았거나, 더 많은 개수가 들어오게 되었을 경우, ErrorCode를 출력하도록 되어 있기 때문에 인수 확인 비교문 수행 후 올바르지 않을 경우 ErrorCode 출력으로 이어진다.

LOAD 명령의 경우, SubtitleQueue의 insert명령을 반복적으로 수행하게 된다. Manager 내부에서 명령 데이터를 받아 SubtitleQueue의 insert명령에 매개변수로 자막 데이터를 넣게 된다. 입력파일에 남은 내용이 없어질 때까지 반복하는 것을 while container를 통해 알 수 있다.

QPOP 명령의 경우, SubtitleQueue와 SubtitleBST와 이어지도록 되어 있다. 만약 큐가 비어있는 상태 또는 SubtitleBST가 이미 내용이 있는 상태라면 예외처리가 되도록 되어 있다. 기본적인 예외 상황 확인 이후에는 SubtitleQueue의 Pop명령을 사용해 받아온 자막 정보를 SubtitleBST의 insert 명령을 통해 삽입하게 된다.

SubtitleBST의 insert명령의 경우, root를 현재 Node로 시작하여, Node가 NULL일 때까지 Position 탐색을 수행한다. 만약 현재 Node의 시간이 더 큰 경우, 왼쪽 Node로 이동하고, 아닌 경우 오른쪽 Node로 이동한다. 중복 시간이 나오는 경우는 프로젝트 전제에서 없다고 정하고 있으므로, 고려하지 않아도 된다.
NULL인 포인터를 찾았다면 해당 Node에 SubtitleBSTNode를 추가한다. 이 Node에는 선언자 실행과 함께 자막 데이터가 들어가게 된다.

PRINT 명령의 경우, SubtitleBST와 SectionList와 이어지게 된다. 인수에 따라서 작동이 변하게 되는데, 인수 길이가 다를 수 있다는 점으로 인해 인수 확인 과정에서 인수 길이를 1보다 큰 경우 Error Code를 출력하도록 되어 있다.
만약 인수 길이가 0이라면 BST 전체를 출력하도록 하는 명령이므로, BST가 비어있는지 확인하고 비어있다면 Error Code, 아니라면 SubtitleBST의 print 멤버 함수를 실행한다. 이 멤버 함수는 root Node부터 하여 재귀적으로 왼쪽 Node에서 Node 출력 함수, 자신 데이터 출력, 오른쪽 Node에서 Node 출력 함수를 실행한다. 따라서, 오름차순으로 자막 데이터를 출력한다.
만약 인수 길이가 1이라면 SectionList에서 특정 Section을 찾아 출력하는 것이기 때문에 SectionList의 search 함수를 실행하고, SectionListNode를 가져온다. 해당 Node의 아래에 있는 SubtitleList를 순회하며 출력하는 것은 Manager 객체 안에서 수행하게 된다.

DELETE 명령의 경우, SubtitleBST의 Node를 삭제하는 기능을 수행한다. 첫 번째 인수에 따라 기능이 달라지는데 UNDER인 경우, SubtitleBST의 delUnder, EQUAL인 경우, SubtitleBST의 delOne을 실행하게 된다. delUnder은 가장 작은 Node를 찾아 해당 Node의 값으로 delOne을 수행하는 것을 반복하도록 하고, delOne은 특정 시각의 Node를 찾아 해당 Node의 자식 수에 따라 삭제 과정을 실행한다. 자식이 없는 경우 부모 Node에서 해당 Node 연결을 끊어주고, 하나인 경우에는 부모 Node와 자신과의 연결을 자신의 자식 Node와의 연결로 바꾸게 된다. 자식 Node가 둘인 경우에는 과제에 제시된 기준에 따라 오른쪽 Node에서 가장 작은 Node를 찾아 자신의 데이터와 그 Node의 데이터를 바꾸고 그 Node에 대해 delOne을 수행한다.

EXIT 명령은 exit 함수를 통해 프로그램을 종료하여 강제로 메모리 해제를 하고 종료한다.

Algorithm

SubtitleQueue에 있어 원형 큐 자료구조를 이용하였다. 프로그램에서는 자막 시간이 중복되지 않으며, 자막 내용은 공백문자 포함하여 200자 이하의 길이를 갖는다고 하였는데, 이는 제한사항이 아닌 구현에 있어 편의 제공이라고 판단하여 본 프로그램에서는 string 헤더를 이용하여 구현에 문제가 없도록 하였다. Queue의 길이 제한에 관련하여 연결 리스트와 비슷한 형태로 Queue를 구현하게 될 경우에는 크기 제한을 확인할 때 전체를 순회하며 크기를 확인하거나, 미리 크기를 기록하는 변수를 만들어둘 필요가 있는 반면에, 원형 큐로 구현하게 되면 길이 제한을 쉽게 설정할 수 있게 된다.

원형 큐에서 빈 큐와 포화 큐를 확인하기 위한 방법은 기억된 front와 rear의 위치를 비교하는 방법으로 가능하게 된다. 원형 큐가 비어있을 경우 front와 rear가 한 자리에 있도록 하여 front와 rear가 같다면 빈 큐라고 확인할 수 있다. 원형 큐가 가득차있을 때는 front가 rear보다 1 많도록 한 상황으로 본다. 단, 이렇게 구현하게 될 경우, 한 칸이 비어있는 상태로 만들어지게 되는데, 원형 큐의 메모리 할당 시 크기를 구현하고자 하는 원형 큐의 크기의 1 더 많은 수치로 선언하면 문제가 해결된다. Push할 경우에는 rear를 하나 밀고 그 자리에 데이터를 추가하며, pop할 경우에는 front를 하나 밀고 그 자리의 데이터를 꺼내면 일반적인 큐와 같이 기능할 수 있다. 추가적으로 원형 큐의 경우 front와 rear수치가 계속 증가하면서 배열의 인덱스와 맞지 않게 되는 경우가 발생하는데, 이는 front와 rear 수치가 변화할 때마다 나머지 연산을 수행하여 특정 값을 넘어가는 경우 0으로 돌아가게 하여 쉽게 해결할 수 있다.

QPOP 명령에서는 IsEmpty가 될 때까지 Pop을 수행하여 전체를 BST로 옮길 수 있도록 하였다.

각 자막에 LOAD에 있어서는 substr을 사용하였다. 시각 정보가 앞으로 오고 00:00:00의 형태를 유지하고 있기에 길이가 일정한 데이터에 대해 쉽게 자막 정보와 시각 정보를 분리할 수 있다.

SubtitleBST에 있어서는 이진 탐색 트리의 구현이 요구사항이다. 본 프로그램에서는 이진 탐색 트리의 노드 위치 기준은 자막시간을 기준으로 하고 있다. 자막 시간을 기준으로 비교연산을 수행할 경우가 많은데 특별한 비교 연산을 수행하기 보다. 자막 시간 비교 시에는 초로 만들어 비교연산을 수행하도록 하였다.

삽입 함수는 while문을 통해 현재 확인하고 있는 node가 NULL포인터일 때까지 진행하고 해당 위치에 자막 데이터를 가진 Node를 추가한다. 따라서, 항상 데이터 위치를 이동하기 전 현재 위치를 parent로 설정하고 빈 Node가 적절한 자리로 확인되었을 경우, parent와 새로운 Node를 이어주게 된다.

출력 함수는 재귀적으로 작동하도록 구현하였다. Node를 기준으로 출력 함수를 실행하면 각 Node에 대해 NULL이 아니라면 왼쪽 Node에 대해 출력 함수를 실행하고 자신의 데이터를 출력한 후 오른쪽 Node에 대해 출력 함수를 실행한다. 이를 통해 In-Order 방식으로 출력을 진행할 수 있도록 하였다. BST에서 print를 수행하면 root Node에서 출력 함수를 실행하게 된다.

탐색 함수 또한 재귀적으로 작동하도록 구현하였다. 찾고자 하는 자막 시간을 기준으로 현재 Node의 자막 시간이 더 이를 경우 오른쪽 Node에서 해당 함수를 다시 실행하는 구조로 되어 있

다. 원하는 Node에 도착하게 되면 해당 값을 반환하고 이 반환이 이어져 처음 함수의 반환 데이터까지 이동하게 된다.

삭제 함수도 마찬가지고 재귀적으로 구현하였다. 자식 Node를 두 개 가지고 있는 Node를 삭제하게 될 경우, 오른쪽 서브트리에서 최솟값을 가진 Node가 현재 Node의 위치로 오게 되는 코드를 작성하여야 하기에, 재귀적으로 구현하지 않아도 데이터 위치를 찾아 구현이 가능하긴 하나, 오른쪽 서브트리만 가지고 있는 Node에 대한 삭제 코드가 두 번 작성되며, 코드가 길어지는 문제가 있어 재귀적인 함수의 구현으로 변경하였다.

정확히는 Node를 찾는 과정에 있어서 재귀적으로 작동하는 것이 대부분이고, Node를 삭제하는 과정에서는 양쪽 서브트리를 모두 가지고 있는 Node에 대해서 삭제 함수가 한 번 더 호출되게 된다.

Search와 같은 과정으로 삭제하고자 하는 자막 시간을 재귀적으로 찾아가 이를 target으로 설정한다. 재귀적으로 호출되는 함수는 항상 자신을 parent 매개변수에 넣어 실행되어 삭제하더라도 parent를 참조할 수 있도록 되어 있다.

삭제 과정은 삭제하고자 하는 Node가 가지고 있는 자식 Node의 개수에 따라 달라지게 된다. 만약 해당 Node가 자식 Node를 가지지 않는다면 parent의 어느 쪽에 자신이 있는 지 확인하여 해당 부분을 nullptr로 바꾸어준다.
만약 해당 Node가 자식 Node를 하나 가지고 있다면 parent의 어느 쪽에 자신이 있는 지 확인하여 해당 부분을 자신의 자식 Node로 바꾸어주게 된다. 만약 parent가 없는 경우 이는 root Node인 경우이므로, parent와의 연결을 변경하는 과정 대신 BST의 root Node를 자식 Node로 설정해주는 과정을 거쳐 오류가 발생하지 않도록 한다.
만약 해당 Node가 자식 Node를 둘 가지고 있다면 해당 Node와의 연결을 조정하는 것보다 값을 대체하는 것이 간편하게 작동하므로, 대체될 값을 가진 Node를 먼저 찾는다. 오른쪽 서브트리에서 가장 작은 값을 가진 Node를 찾아야 하는데, 계속 왼쪽 자식으로 가면 이를 찾을 수 있다. 가장 왼쪽이므로, 왼쪽 서브트리가 없음이 보장되므로, 오른쪽 서브트리의 유무에 따라 삭제 과정을 진행할 수 있다. 이에 대해서는 현재 삭제 대상인 Node부터 해당 오른쪽 자식 방향으로 대상만 대체 데이터를 가진 Node로 잡고 재귀적인 삭제 함수를 다시 수행시작한다. 이후 해당 Node의 데이터는 남아있고 연결만 사라지기에 해당 Node의 데이터를 삭제하고자 하는 Node의 데이터에 덮어씌워주는 방식으로 구현하였다.

특정 시각 이전의 시간 정보를 가지는 자막을 한 번에 삭제하는 경우, 가장 작은 시간 정보를 가진 Node를 반복적으로 호출하여 해당 Node가 매개변수로 제공된 시각 이전일 경우, 삭제하고 매개변수로 제공된 시각 이후인 Node가 나올 때까지 반복한다.
지정한 시각과 가장 가까운 왼쪽 Node를 찾는 방법도 존재할 것은 같으나, 데이터의 삭제가 제대로 수행되지 않거나 하는 등의 문제가 발생할 우려가 있다 판단하여 메모리 누수를 방지하고자 단순하고 확실한 방법으로 작동하도록 구현하였다.

섹션 연결 리스트의 경우, 섹션을 잇는 연결 리스트 하나와 각 섹션에 들어가는 자막 정보를 나타내는 자막 연결 리스트의 두 리스트를 기준으로 하는 2차원 연결 리스트로 스켈레톤 코드가 제

공되었다. 따라서, 해당 방식에 적절하도록 구현하였다.

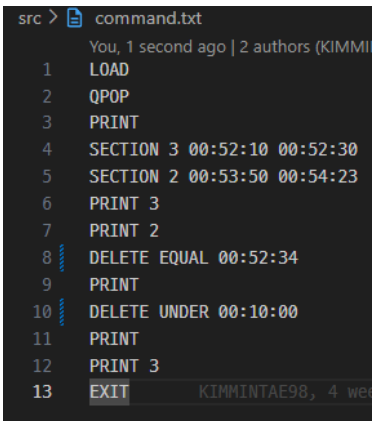
섹션을 잇는 것은 단순히 연결 리스트에 next만 새로 지정해주는 방식으로 구현하였고, 섹션에 데이터를 추가하는 구현에 있어 BST의 root Node와 함께 재귀적 방식으로 값을 가져오도록 하였다.

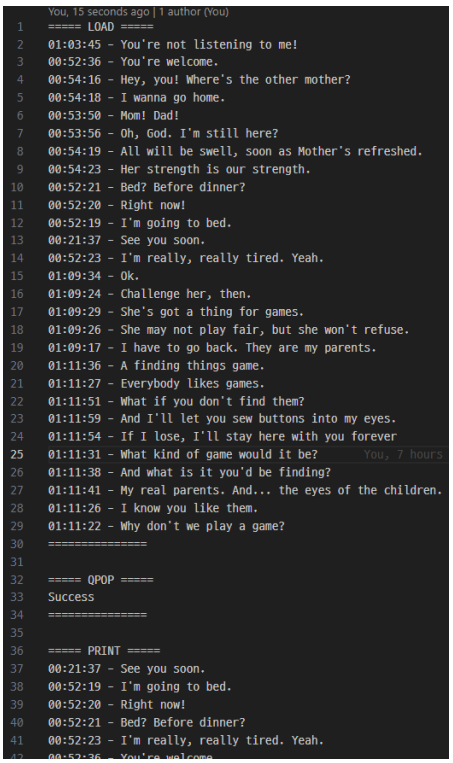
BST에서 출력기능을 사용하듯, In-Order 순회로 왼쪽 Node부터 지정한 섹션의 범위에 들어가는지 확인하고 해당 범위에 들어간다면 SubtitleListNode에 insert되도록 하였다. 만약 범위에 들어가지 않는다면 그 값이 범위에서 앞으로 벗어났는지 뒤로 벗어났는지 확인하여 각각 왼쪽 Node, 오른쪽 Node로는 더 이상 탐색하지 않도록 하여 시행 횟수를 감소시켰다.

SectionListNode가 만들어지면 해당 Node내부의 자막 데이터 개수를 확인하여 만약 개수가 0이라면 오류를 반환하도록 하였다.

Result Screen

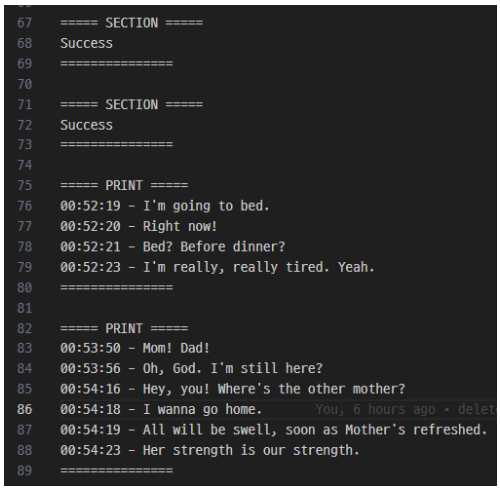
입출력 예시에서 살짝 바뀐 입력이다. LOAD QPOP PRINT 부분은 오른쪽 출력 사진과 같이 정상적으로 출력됨을 볼 수 있다.

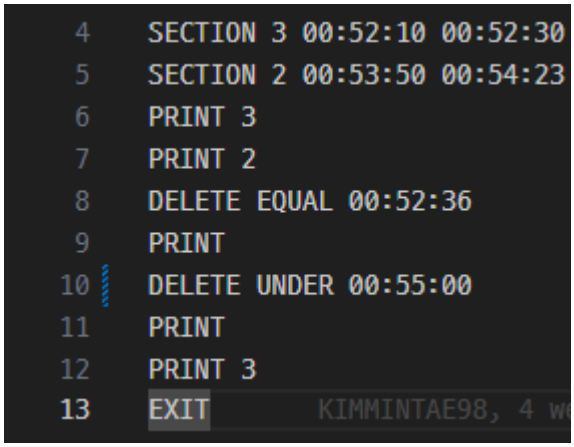




섹션 명령어와 섹션 부분 출력 명령어 또한 정상작동함을 오른쪽 아래 사진을 통해 확인할 수 있다.

섹션 부분으로 지정한 범위 내의 자막만 들어감이 출력을 통해 확인된다.





명령어를 왼쪽과 같이 변경하고 삭제 명령에 대한 테스트를 입출력 결과를 확인하였다.

```
===== DELETE =====
Success
==============

===== PRINT =====
00:21:37 - See you soon.
00:52:19 - I'm going to bed.
00:52:20 - Right now!
00:52:21 - Bed? Before dinner?
00:52:23 - I'm really, really tired. Yeah.
00:53:50 - Mom! Dad!
00:54:16 - Hey, you! Where's the other mother?
00:54:18 - I wanna go home.
00:54:19 - All will be swell, soon as Mother's refreshed.
00:54:23 - Her strength is our strength.
01:03:45 - You're not listening to me!
01:09:17 - I have to go back. They are my parents.
01:09:24 - Challenge her, then.
01:09:26 - She may not play fair, but she won't refuse.
01:09:29 - She's got a thing for games.
01:09:34 - Ok.
01:11:22 - Why don't we play a game?
01:11:26 - I know you like them.
01:11:27 - Everybody likes games.
01:11:31 - What kind of game would it be?
01:11:36 - A finding things game.
01:11:38 - And what is it you'd be finding?
01:11:41 - My real parents. And... the eyes of the children.
01:11:51 - What if you don't find them?
01:11:54 - If I lose, I'll stay here with you forever
01:11:59 - And I'll let you sew buttons into my eyes.
==============
```

삭제 명령을 수행한

00:52:36 - You're welcome.

문장이 사라져있음을 확인할 수 있다.

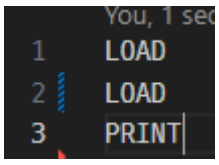
```
===== DELETE =====
Success
==============

===== PRINT =====
01:03:45 - You're not listening to me!
01:09:17 - I have to go back. They are my parents.
01:09:24 - Challenge her, then.
01:09:26 - She may not play fair, but she won't refuse.
01:09:29 - She's got a thing for games.
01:09:34 - Ok.
01:11:22 - Why don't we play a game?
01:11:26 - I know you like them.
01:11:27 - Everybody likes games.
01:11:31 - What kind of game would it be?
01:11:36 - A finding things game.
01:11:38 - And what is it you'd be finding?
01:11:41 - My real parents. And... the eyes of the children.
01:11:51 - What if you don't find them?
01:11:54 - If I lose, I'll stay here with you forever
01:11:59 - And I'll let you sew buttons into my eyes.
==============
```

두 번째 삭제 명령으로는 55초 이전의 자막들을 모두 삭제하도록 하였는데, 모두 정상적으로 삭제되었음을 확인할 수 있다.

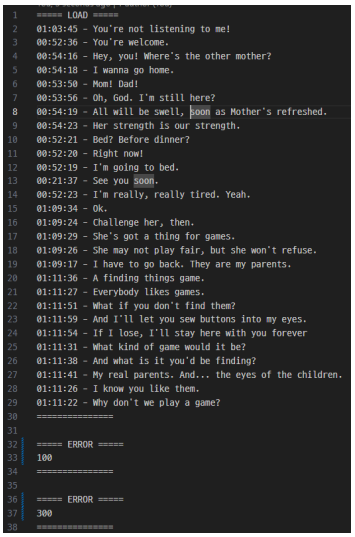
섹션은 따로 작동하기에 섹션 3의 자막들은 남아있음을 볼 수 있다.

```
===== PRINT =====
00:52:19 - I'm going to bed.
00:52:20 - Right now!
00:52:21 - Bed? Before dinner?
00:52:23 - I'm really, really tired. Yeah.
==============
```



명령어를 왼쪽과 같이 단 3줄만 작성하였다.

출력 결과는 오른쪽과 같이 첫 LOAD만 정상적으로 이루어지고 예외 처리 조건에 의해 LOAD와 PRINT는 오류가 나게 된다.



```
===== LOAD =====
01:03:45 - You're not listening to me!
00:52:36 - You're welcome.
00:54:16 - Hey, you! Where's the other mother?
00:54:18 - I wanna go home.
00:53:50 - Mom! Dad!
00:53:56 - Oh, God. I'm still here?
00:54:19 - All will be swell, soon as Mother's refreshed.
00:54:23 - Her strength is our strength.
00:52:21 - Bed? Before dinner?
00:52:20 - Right now!
00:52:19 - I'm going to bed.
00:21:37 - See you soon.
00:52:23 - I'm really, really tired. Yeah.
01:09:34 - Ok.
01:09:24 - Challenge her, then.
01:09:29 - She's got a thing for games.
01:09:26 - She may not play fair, but she won't refuse.
01:09:17 - I have to go back. They are my parents.
01:11:36 - A finding things game.
01:11:27 - Everybody likes games.
01:11:51 - What if you don't find them?
01:11:59 - And I'll let you sew buttons into my eyes.
01:11:54 - If I lose, I'll stay here with you forever
01:11:31 - What kind of game would it be?
01:11:38 - And what is it you'd be finding?
01:11:41 - My real parents. And... the eyes of the children.
01:11:26 - I know you like them.
01:11:22 - Why don't we play a game?
==============

===== ERROR =====
100
==============

===== ERROR =====
300
==============
```

```
LOAD
SECTION 1 00:00:00 00:10:00
QPOP
SECTION 1 00:00:00 00:10:00
SECTION 2 00:50:00 00:53:00
PRINT 1
PRINT 2
PRINT 3
EXIT
```

섹션 데이터의 저장과 출력 확인을 위해 명령어를 왼쪽과 같이 작성하였을 경우,

```
===== ERROR =====
400
==============

===== QPOP =====
Success
==============

===== ERROR =====
400
==============

===== SECTION =====
Success
==============

===== ERROR =====
300
==============

===== PRINT =====
00:52:19 - I'm going to bed.
00:52:20 - Right now!
00:52:21 - Bed? Before dinner?
00:52:23 - I'm really, really tired. Yeah.
00:52:36 - You're welcome.
==============

===== ERROR =====
300
==============

===== EXIT =====
Success
==============
```

오른쪽과 같이 첫 섹션은 BST가 비어 에러, QPOP이 후에는 들어갈 수 있는 자막 데이터가 없어 에러가 난다. 해당되는 자막이 있도록 범위를 바꾸어 다른 섹션 생성 시 정상작동한다. 이후 출력에서도 1번은 생성 예외로 인해 생성 자체가 되지 않았으므로 에러, 2번은 생성되었으니 출력, 3번은 생성된 적이 없으므로 출력되지 않는다.

```
LOAD
QPOP
DELETE EQUAL 00:10:00
DELETE UNDER 00:10:00
DELETE UNDER 01:00:00
PRINT
EXIT
```

삭제 명령의 확인을 위해 왼쪽과 같이 명령어를 작성하고 테스트를 수행하였다.

```
===== ERROR =====
500
==============

===== ERROR =====
500
==============

===== DELETE =====
Success
==============

===== PRINT =====
01:03:45 - You're not listening to me!
01:09:17 - I have to go back. They are my parents.
01:09:24 - Challenge her, then.
01:09:26 - She may not play fair, but she won't refuse.
01:09:29 - She's got a thing for games.
01:09:34 - Ok.
01:11:22 - Why don't we play a game?
01:11:26 - I know you like them.
01:11:27 - Everybody likes games.
01:11:31 - What kind of game would it be?
01:11:36 - A finding things game.
01:11:38 - And what is it you'd be finding?
01:11:41 - My real parents. And... the eyes of the children.
01:11:51 - What if you don't find them?
01:11:54 - If I lose, I'll stay here with you forever
01:11:59 - And I'll let you sew buttons into my eyes.
==============
```

첫 두 번의 삭제에 대해서 삭제할 항목이 하나도 존재하지 않기에 오류가 나고, 세 번 째 삭제 명령은 삭제할 대상이 존재하기에 성공적으로 삭제가 된다. 이후 BST 전체 출력에 대해서도 삭제된 상태로 정상적으로 출력이 됨을 볼 수 있다.

Consideration

먼저 큐 자료구조를 작성할 시기에 처음에는 연결 리스트와 비슷한 방식으로 큐를 구현하여 사용하였으나, 자막 큐의 크기 제한 등의 제한 사항을 확인한 후, 원형 큐가 더 구현하기에 좋음을 알고 해당 방식으로 코드를 개선하였다. 크기 제한이 없이 사용하기 위한 연결 리스트 방식의 큐가 아닌 원형 큐로 현재 노드의 개수를 셀 필요 없이 front와 rear의 단순한 비교로 코드 작동이 가능함에 코드의 단순함을 챙길 수 있었다. 빈 큐와 포화 큐에 대해 처리하는 코드의 경우, 구현 당시에 떠올리지 못해 인터넷에서 원형 큐 작성에 대한 구조를 참고하여 작성하였다.

BST 작성 당시에는 삽입부터 여러 구현에 있어 비교대상인 자막 시간을 적절히 비교할 방법이 필요했는데, 시분초를 쉽게 비교하기 위해 해당 수치들을 초로 통합하여 계산하는 방식을 떠올리고, 해당 방식을 좀 더 쉽게 적용하기 위하여 저장할 때, Subtitle이라는 클래스를 이용하여 저장하도록 하였다. 해당 클래스는 자막 정보와 자막 시간 정보를 저장하는데 출력의 편의성을 위하여 시/분/초를 따로 저장하지만, 필요에 따라 toSeconds 멤버 함수를 통해 이들을 초로 통합한 수치를 꺼낼 수 있도록 하여 데이터의 호출과 비교에 있어 편하게 구현할 수 있도록 하였다.

이는 BST의 삽입부터 탐색 삭제까지 모든 부분에 있어 편의성을 제공할 수 있었고, 특히 삭제 연산에서 다양한 데이터들을 일일이 옮기지 않더라도 노드간의 데이터 교환을 Subtitle 클래스 데이터 하나의 교환으로 이루어낼 수 있어 코드 관리에 상당한 편의를 가질 수 있게 되었다.

BST의 삽입, 탐색 로직에 대해서는 큰 고민이 없었으나, 삭제 함수에 대해서는 한 번 큰 수정을 거친 바가 있다. 기존에는 재귀적인 방식이 아닌, 삭제 대상을 찾아 자식 노드 조건 확인 후 바로 삭제를 수행하도록 하였으나, 자식 노드를 2개 가지고 있는 경우의 코드가 삭제 코드를 두 번 거치게 되기에 코드가 다소 길어지고 코드 관리에 용이하지 않은 문제가 있어 재귀적 방식의 삭제로 바꾸게 되었다. 이에 따라 여러 개의 자식 노드를 가지고 있는 경우에 우측 서브트리의 최소값과 값 교환 이후에 삭제 작업을 진행할 수 있도록도 바뀌어 코드관리의 편의성을 챙길 수 있었다.

UNDER 삭제의 경우에는 원래 EQUAL과 완전히 다른 로직을 적용하고자 시도가 있었다. 해당 기준 이하의 최대 수치를 가진 노드를 찾아 노드간의 연결을 정리하는 방식을 사용하고자 생각은 했었으나, root가 바뀌는 경우가 생기는 것을 막기 힘들고, Node아래에서의 서브트리들을 정리할 때 충분히 큰 값이 있음을 인지하지 못하고 계속 추가적인 삭제나 모자란 삭제가 발생하여 단순하지만 확실한 방법인 최소값을 계속 체크하면서 삭제하는 방식을 택하게 되었다. 이후에 이 부분에 대해 생각해볼 시간이 생긴다면 다시 한 번 시도해볼 생각이다.

프로젝트 전체적인 부분에 있어 구현에는 편의성의 문제를 제외하고는 큰 어려움없이 구현한 부분이 있었는데, 단지 컴파일러가 Ubuntu에서 사용하는 gcc로 제한되기에 이를 사용하는데 다소 어려움이 있었다. 처음 사용하는 Linux 운영체제에서 사용법을 익히기 위해 인터넷 서핑을 많이 진행하였다.
결국 Windows에서 개발 후 gcc 컴파일러를 Windows에도 설치해 기본적인 디버깅을 진행하고, Ubuntu에서 정상 작동확인을 위해 Makefile을 이용한 컴파일 후 작동 재확인하는 과정을 거쳐야 했다.

또, 기본적으로 스켈레톤 코드에 있는 파일이 상당히 많았고, h와 cpp로 분리된 것이 많아 컴파일 과정에서 이 둘을 한 번에 수정하지 않아 생기는 작은 오류들이 많이 발생하였다. 이번 프로젝트를 진행하며 이런 부분에 있어 계속 확인하는 것이 오류 확인 탭을 줄일 수 있는 중요한 방식이라는 것을 알 수 있었다.

헤더 파일의 반복 불러오기를 방지하기 위한 pragma once의 경우에도 이번에 알게 되었다. ifndef를 이용한 방식은 기존에도 알고 있었는데, 짧은 구문을 통해 이를 이용할 수 있는 것을 알게 되어 코드 관리를 더 편하게 할 수 있는 방법을 알게 되었다.
단지, 해당 구문을 사용하게 될 경우, Makefile에서 계속 pragma once 관련 warning을 띄우기에 컴파일 시간에 영향을 주지는 않을 지 걱정되는 부분이 있다.